

**APPENDIX A**

```

5  #ifndef IO_H
   #define IO_H
   #include <stdio.h>

   /* $Id: io.h,v 1.1 2000/07/11 20:06:20Z drh Exp drh $ */

   #define IO_T T
   typedef struct T *T;

10  extern T IO_open(const char *file, const char *mode);
   extern int IO_close(T stream);
   extern int IO_flush(T stream);
   extern int IO_getc(T stream);
15  extern int IO_putc(int c, T stream);
   extern int IO_read(char *ptr, size_t size, size_t count, T stream);
   extern int IO_write(char *ptr, size_t size, size_t count, T stream);

   extern T IO_stdin;
20  extern T IO_stdout;
   extern T IO_stderr;

   #undef T
   #endif

25

   #define IO_T T

   /* Standard file I/O */

30  struct file {
       struct T stream;
       FILE *fp;
   };

35  static int fileclose(T stream) {
       FILE *fp = ((struct file *)stream)->fp;
       return fclose(fp);
   }

40  static int fileflush(T stream) {
       FILE *fp = ((struct file *)stream)->fp;
       return fflush(fp);
   }

45  static int fileread(char *ptr, size_t size, size_t count, T stream) {
       FILE *fp = ((struct file *)stream)->fp;
       return fread(ptr, size, count, fp);
   }

50  static int filewrite(char *ptr, size_t size, size_t count, T stream)
   {

```

```

        FILE *fp = ((struct file *)stream)->fp;
        return fwrite(ptr, size, count, fp);
    }

5   static struct methods fileio = { fclose, fflush, fread,
    fwrite };

    static T fopen(const char *file, const char *mode) {
        FILE *fp = fopen(file, mode);
10        if (fp) {
            struct file *stream = malloc(sizeof *stream);
            if (stream) {
                stream->stream.methods = &fileio;
                stream->fp = fp;
15                return (T)stream;
            }
            fclose(fp);
        }
        return NULL;
20    }

    static struct file
        stdin  = { &fileio, stdin },
        stdout = { &fileio, stdout },
25        stderr = { &fileio, stdout };
    T IO_stdin = (T)&stdin, IO_stdout = (T)&stdout, IO_stderr =
        (T)&stderr;

```

## APPENDIX B

```

30    /* Net I/O */

    #ifdef WIN32
    #include <windows.h>
35    #include <wininet.h>

    static HINTERNET hSession = NULL;

    struct net {
40        struct T stream;
        HINTERNET hFile;
        char buffer[128];
        char *bp, *limit;
    };

45    static void netcleanup(void) {
        if (hSession)
            InternetCloseHandle(hSession);
        hSession = NULL;
50    }

    static int netclose(T stream) {
        HINTERNET hFile = ((struct net *)stream)->hFile;
        return InternetCloseHandle(hFile) == TRUE ? 0 : EOF;

```

```

}

static int netflush(T stream) {
    return EOF;
}

static int netread(char *ptr, size_t size, size_t count, T stream) {
    struct net *ns = (struct net *)stream;
    size_t n = count*size;
    if (ns->bp < ns->limit) {
        for ( ; ns->bp < ns->limit && n >
0; n--)
            *ptr++ = *ns->bp++;
        return (count*size - n)/size;
    }
    if (InternetReadFile(ns->hFile, ptr, n, &count) == FALSE)
        count = 0;
    return count;
}

static int httpError(struct net *stream) {
    int count;
    char *bp = stream->bp = stream->limit = stream->buffer;
    if (!InternetReadFile(stream->hFile, stream->bp, sizeof
stream->buffer, &count))
        return 0;
    stream->limit = stream->buffer + count;
    for ( ; bp < stream->limit; bp++)
        if (*bp == '<' && (strncmp(bp, "<title>", 7)
== 0 || strncmp(bp, "<TITLE>", 7) == 0)) {
            int code = 0;
            for (bp += 7; bp < stream->limit
&& isspace(*bp); )
                bp++;
            while (bp < stream->limit &&
                code = 10*code +
                if (code >= 401 && code <= 505)
                    return 1;
            return 0;
        }
    return 0;
}

static T netopen(const char *file, const char *mode) {
    static struct methods netio = { netclose, netflush,
netread, nullwrite };
    HINTERNET hFile;
    if (hSession == NULL) {
        hSession = InternetOpen("",
INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
        if (hSession);
        atexit(netcleanup);
    }
}

static T netopen(const char *file, const char *mode) {
    static struct methods netio = { netclose, netflush,
netread, nullwrite };
    HINTERNET hFile;
    if (hSession == NULL) {
        hSession = InternetOpen("",
INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
        if (hSession);
        atexit(netcleanup);
    }
}

```

```

    }
    if (strspn(mode, "RrbB") != strlen(mode))
        return NULL;
    hFile = InternetOpenUrl(hSession, file, NULL, 0, 0, 0);
5   if (hFile) {
        struct net *stream = malloc(sizeof *stream);
        if (stream) {
            stream->stream.methods = &netio;
            stream->hFile = hFile;
10          if (httpError(stream) == 0)
                return (T)stream;
            IO_close((T)stream);
            return NULL;
        }
15        InternetCloseHandle(hFile);
    }
    return NULL;
}
#else
20
static T netopen(const char *file, const char *mode) {
    return NULL;
}
#endif
25
int IO_close(T stream) {
    int code;
    assert(stream);
    code = (*stream->methods->close)(stream);
30    free(stream);
    return code;
}

int IO_flush(T stream) {
35    assert(stream);
    return (*stream->methods->flush)(stream);
}

int IO_getc(T stream) {
40    char c;
    assert(stream);
    if ((*stream->methods->read)(&c, 1, 1, stream) == 1)
        return (unsigned)c;
    return EOF;
45 }

int IO_putc(int c, T stream) {
    char buf = c;
    assert(stream);
50    if ((*stream->methods->write)(&buf, 1, 1, stream) == 1)
        return c;
    return EOF;
}

```

```

int IO_read(char *ptr, size_t size, size_t count, T stream) {
    assert(ptr);
    assert(stream);
    return (*stream->methods->read)(ptr, size, count,
5    stream);
}

```

```

int IO_write(char *ptr, size_t size, size_t count, T stream) {
    assert(ptr);
    assert(stream);
    return (*stream->methods->write)(ptr, size, count,
10    stream);
}

```

```

15 static int isUrl(const char *path) {
    return strstr(path, "://") != NULL;
}

```

```

20 T IO_open(const char *file, const char *mode) {
    const char *s;
    assert(mode);
    for (s = mode; *s; s++)
        if (strchr("AaBbRrWw+", *s) == NULL)
25         return NULL;

    if (file == NULL)
        return nullopen(file, mode);
    else if (isUrl(file))
        return netopen(file, mode);
30    else
        return fileopen(file, mode);
}

```